

500.42884X00

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

Applicant(s): T. ENDO, et al  
Serial No.: 10/608,209  
Filed: June 30, 2003  
Title: INFORMATION PROCESSING MEANS

**RECEIVED**

**MAR 03 2004**

Technology Center 2100

**LETTER CLAIMING RIGHT OF PRIORITY**

Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

August 18, 2003

Sir:

Under the provisions of 35 USC 119 and 37 CFR 1.55, the applicant(s) hereby  
claim(s) the right of priority based on:

**Japanese Patent Application No. 2003-014136**  
**Filed: January 23, 2003**

A certified copy of said Japanese Patent Application is attached.

Respectfully submitted,

**ANTONELLI, TERRY, STOUT & KRAUS, LLP**

\_\_\_\_\_  
Carl I. Brundidge  
Registration No. 29,621

CIB/rp  
Attachment

日 本 国 特 許 庁  
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日

Date of Application: 2003年 1月23日

出 願 番 号

Application Number: 特願2003-014136

[ST.10/C]:

[JP2003-014136]

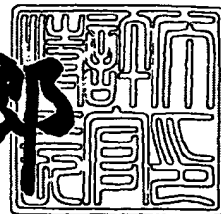
出 願 人

Applicant(s): 株式会社日立製作所

2003年 6月10日

特 許 庁 長 官  
Commissioner,  
Japan Patent Office

太田 信一郎



出証番号 出証特2003-3044975

【書類名】 特許願  
【整理番号】 H02016081A  
【あて先】 特許庁長官 殿  
【国際特許分類】 H04L 9/10

【発明者】

【住所又は居所】 東京都国分寺市東恋ヶ窪一丁目280番地 株式会社日立製作所中央研究所内

【氏名】 遠藤 隆

【発明者】

【住所又は居所】 東京都国分寺市東恋ヶ窪一丁目280番地 株式会社日立製作所中央研究所内

【氏名】 神永 正博

【発明者】

【住所又は居所】 東京都国分寺市東恋ヶ窪一丁目280番地 株式会社日立製作所中央研究所内

【氏名】 渡邊 高志

【特許出願人】

【識別番号】 000005108

【氏名又は名称】 株式会社 日立製作所

【代理人】

【識別番号】 100075096

【弁理士】

【氏名又は名称】 作田 康夫

【電話番号】 03-3212-1111

【手数料の表示】

【予納台帳番号】 013088

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1  
【物件名】 要約書 1  
【ブルーフの要否】 要

【書類名】 明細書

【発明の名称】 情報処理方法

【特許請求の範囲】

【請求項1】

^ をベキ乗演算と定義し、Pを素数とし、 $x > P$ であるxに対して、 $x * (2^n) \bmod P$ を計算する情報処理方法に於いて、法をP、入力値をx、法Pを格納するために必要十分なビット数をn、入力値xを格納するために必要なビット長をmとし、入力値を $x * (2^n) \bmod P$ に変換する際に、 $2^{(2m+n)} \bmod P$ を計算するか、あるいは予め用意しておき、モンゴメリ剰余乗算により $x_1 = x * 2^{(2m+n)} * (2^{-m}) \bmod P = x * 2^{(m+n)} \bmod P$ を計算し、さらに $x_2 := x_1 * (2^{-m}) \bmod P = x * (2^n) \bmod P$ を計算することにより、 $x \bmod P$ を陽に求めること無しに $x * (2^n) \bmod P$ を計算することを特徴とする情報処理方法。

【請求項2】

^ をベキ乗演算と定義し、Pを素数とし、 $x > P$ であるxに対して、 $x * (2^n) \bmod P$ を計算する情報処理方法に於いて、法をP、入力値をx、法Pを格納するために必要十分なビット数をn、入力値xを格納するために必要なビット長をmとし、入力値を $x * (2^n) \bmod P$ に変換する際に、 $2^{(m+2n)} \bmod P$ を計算するか、あるいは予め用意しておき、モンゴメリ剰余乗算方式により $x_1 = x * 2^{(m+2n)} * (2^{-m}) \bmod P = x * 2^{(2n)} \bmod P$ を計算し、さらに $x_2 := x_1 * (2^{-n}) \bmod P = x * (2^n) \bmod P$ を計算することにより、 $x \bmod P$ を陽に求めること無しに $x * (2^n) \bmod P$ を計算することを特徴とする情報処理方法。

【請求項3】

法をP、入力値をx、法Pを格納するために必要十分なビット数をn、入力値xを格納するために必要なビット長をm、指数をdとし、ベキ乗の指数をsビットづつとりだしてsビット毎のベキ乗演算の結果を合成して、 $x^d \bmod P$ のベキ乗剰余演算を行う情報処理方法において、sビットづつ取り出した指数のi番目の指数を $d[i]$ としたとき、 $x^d \bmod P$ を演算する代わりに $(2^n$

$(2^s - 1) * x^d[i] \bmod P$ を用いて演算を行い、 $(2^n)^{(2^{n-1}) * x^d \bmod P}$ を計算した後、 $2^{(-n)^{(2^{n-1}) \bmod P}}$ を乗じて、 $x^d \bmod P$ を計算することを特徴とする情報処理方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は情報処理方法に関し、特に例えば、機密性の高いICカードなどの耐タンパ装置に関するものである。

【0002】

【従来の技術】

RSAの高速演算手法のCRT(Chinese Remainder Theory)演算方式では、計算の一番最初のステップで $x \bmod p$ を計算する。CRT演算方式の処理手順を図1に示す。まず、 $p$ で還元した値に対して剰余べき乗演算を行い(1010)、 $q$ で還元した値に対するべき乗剰余演算を行う(1020)。最後に2つのべき乗剰余演算結果を合成し(1030)最終的な結果を得る。最初の $p$ で還元した値の剰余べき乗演算(1010)および $q$ で還元した値のべき乗剰余演算(1020)の最初のステップで、それぞれ秘密指数 $p$ に対する剰余、秘密指数 $q$ に対する剰余を計算する必要がある。べき乗剰余を計算するには、剰余乗算を繰り返し行うことで実現する。

【0003】

1010、1020のべき乗計算には通常、アディションチェイン手法を用いる。アディションチェインとは、たとえば、 $Z=A^L$ を計算する場合、指数 $L$ を二進数展開し、

$$L=L[n-1]*2^{(n-1)}+L[n-2]*2^{(n-2)}+\dots+L[1]*2^1+L[0]*2^0 \quad (式1)$$

と置き、指数の加算は掛算になり、指数の乗算がべき計算になるという指数法則を用いて、 $Z=A^L$ の計算を

$$Z:=\left(\dots\left(\left(A^L[n-1]\right)^2 * \left(A^L[n-2]\right)\right)^2 * \dots * \left(A^L[0]\right)\right) \quad (式2)$$

とおく。 $A^L[i]$ は、 $L[i]=1$ であれば、 $A$ となり、 $L[i]=0$ であれば、 $A^0=1$ となり

、 $L[i]=0$ の場合に1の掛算を省略すると、 $L$ を2進表記した際の1となるビット数回の掛算と、 $n-1$ 回の自乗計算により、 $A^L$ が計算できる。

プログラムにより表現すると、

$Z = 1$

```
for ( i = n-1 ; i >= 0 ; i++ ) {
    W := W * W;
    if ( L[i]==1 ) then Z:=Z * A; else W:=W * 1;
}
```

となる。

【0004】

剰余乗算の方式には大きくモンゴメリ剰余乗算を用いたものと、そうでないものの2つに分けることができる。

【0005】

図2はモンゴメリ剰余乗算を用いた場合のアディションチェインによるべき乗剰余計算の処理フローである。 $n$ は、 $P$ を格納するのに十分なビット長を示す。

まず、 $P$ に対する $x$ の剰余を2020で求める。モンゴメリ剰余乗算では、乗算の度に $2^{(-n) \bmod P}$ が乗ぜられるため、予め $2^n$ を被演算数に乗じておく。以下、 $R=2^n$ とおく。被演算数に $R$ を乗ずるのにも、モンゴメリ剰余乗算を用いる。あらかじめ $R^2$ を計算しておき、モンゴメリ剰余乗算を用いて $x \bmod P$ に乘じ、 $xR \bmod P$ を得る(2040)。モンゴメリ剰余乗算を用いるので、演算の初期値は1の代わりに1に $R$ を乗じた $R$ とする。乗算の処理は、指数の最上位のビットから1ビットずつ取り出すので、カウンタ $i$ に最上位ビットの位置を示す $n-1$ を設定する(2050)。指数上位ビットから順に1が立っているかどうかを調べ(2060)、0であれば、1を意味する $R$ を乗じ(2070)、1が立っているならば、 $xR \bmod P (=AR)$ を $W$ に乘じる(2080)。1を意味する $R$ を乗じる処理2070は、計算結果に影響を与えないので、処理速度を重視する場合は省略可能である。ビットの位置カウンタを1ビット分移動し(2090)、最下位ビットまでたどり着いているかチェックを行い(2100)、最下位ビットまで達していない場合は結果を二乗し(2110)、指数の次のビットに対して3060か

らの処理から処理を繰り返す。2100で最下位ビットまで処理が終了した場合は、 $2^n$ が掛かっている影響を取り除くため、 $2^{-n}$ を乗じる。モンゴメリ剰余乗算で、1との積を計算することは、 $2^{-n}$ を乗じることと等しい(3120)。最後に、結果がP以上の場合は(2130)、Pを減ずる(2140)。一連の処理の中で、2020の剰余演算の結果は、図3に示すように、Pの倍数(3010)を境にPより大であるか小であるかによって、大きく値が変動するため、アタックポイントとなる可能性がある。

## 【0006】

「図4」はモンゴメリではない剰余乗算を用いてアディションチェーンによるべき乗剰余演算を行った場合の処理フローである。Pのビット長をnとおく。つぎに、まずxのPに対する剰余を4020で求める。モンゴメリ剰余乗算を用いた場合の処理と同様、このPに対する剰余を求める演算がアタックポイントとなる可能性がある。通常の剰余乗算を用いるので、演算の初期値は1とし、最上位のビットから1ビットずつ取り出すので、カウンタiに最上位ビットの位置を示すn-1を設定する(4040)。指数上位ビットから順に1が立っているかどうかを調べ(4050)、1が立っていれば、 $x \bmod P (=A)$ をWに乘じ(4070)、ビットの値が0であれば、1を意味を乗じる(4060)。また、1を乗じる処理4060は、計算結果に影響を与えないので、処理速度を重視する場合は省略可能である。4070の処理で $x \bmod P$ の値が被演算数に使用されるので、ここもアタック対象になる可能性がある。ビットの位置を1ビット下に移動し(3080)、最下位ビットまでたどり着いていない場合は、結果を二乗し(3090)、指数の次のビットの処理を行う。最下位ビットまで処理が終了した場合は、その時点でのWが計算結果となる。

## 【0007】

以上のように、モンゴメリ剰余乗算を用いている場合も、通常の剰余乗算を行った場合も、Pによる剰余算結果が処理の最初に必要となりアタックポイントとなる可能性がある。

## 【0008】

【発明が解決しようとしている課題】



RSA暗号は認証や、秘密鍵の配送などに標準的に用いられている暗号で、金融用途等ではその演算の安全性が非常に重要視されている。RSA暗号の高速演算法として、中国人剰余定理を用いた計算手法が広く用いられているが、その一番最初の演算で、秘密素数 $p$ による剰余計算が必要となる。この計算は、秘密素数 $p$ を陽に用いた計算であるので、古くからアタックの対象となっている。 $p$ による剰余計算で問題となるのは、図2に示すように $x$ が $p$ の倍数近傍(2010)の値の場合  $x < kp$  では、 $x \bmod p \approx p$  となり大きな値となる一方、 $x > kp$ の場合は、 $x \bmod p \approx 0$ となり、小さな値となることである。 $x \bmod p$ の値が $p$ を境界として大きく変動するため、入力 $x$ が秘密指数 $p$ よりも大なのか小であるのかが、電流値等のサイドチャネル情報として識別できる危険性がある。RSA暗号では、大きな素数(現在は、512bit程度の素数が用いられている) $p$ 、 $q$ の積 $N$ が容易に因数分解できないことを安全性の根拠としており、 $N$ は公開鍵の一部としてユーザに公開されている。秘密素数 $p$ もしくは $q$ がリークすると、 $N/p$ は容易に計算可能であるので、秘密鍵 $d$ は公開鍵 $e$ の $(p-1)(q-1)$ を法とする逆元を計算することにより、求めることが出来る。逆元計算は、拡張ユークリッド互除法により、容易に計算可能である。本発明は、CRT向けの剰余計算を高速かつ安全に行うための計算方法および装置に関するものである。

【0009】

【課題を解決するための手段】

$x \bmod p$ を直接計算せずに、 $2^m (m+n) \bmod p$ もしくは、 $2^n (2n) \bmod p$ をあらかじめ $x$ に乗じておき、その後、 $2^m (-n)$ もしくは、 $2^n (-m)$ を乗じて、 $2^n x \bmod p$ を計算する。 $p$ は大きな素数であるので常に奇数となり、2のべき乗とは常に互いに素になるので、 $2^m (-m) \bmod p$ もしくは  $2^n (-n) \bmod p$ は必ず存在する。また、 $2^n x \bmod p$ の値は、入力 $x$ が $p$ の近傍にある場合でも $x$ と $p$ の大小関係に依存した値の大きな変化は発生せず、 $2^m x \bmod p$ のビット長は $p$ のビット長に近い値となる。したがって、リーク情報から $x$ と $p$ の大小関係を推定することが不可能となり、秘密鍵の漏洩を防止することを可能とする。モンゴメリ剰余乗算を用いる場合には、 $2^n$ を乗じた形式は、そのままモンゴメリフォーマットになっているので、以降の処理は、従来の処理フローを使用できる。

## 【0010】

モンゴメリ剰余乗算を用いない場合には、指数演算計算の最後に、 $(2^{(-n)})^{(2^{n-1}) \bmod p}$  を乗じ、 $2^n \bmod p$  を乗じた影響を補正することで、正しい結果を得る。

## 【0011】

モンゴメリ乗算以外の剰余乗算を用いる場合は、乗算と自乗を行った後、 $R^{(-2) \bmod p}$  を乗じる。あらかじめ  $R^{(-2m) \bmod p}$  を計算しておき、最後に  $R^{(-(2^n m)+1) \bmod p}$  を乗じて補正してもよい。

## 【0012】

## 【発明の実施の形態】

図5はモンゴメリ剰余乗算を用いた場合の本発明の一実施例を示す。 $m$ を入力  $x$  の格納に必要なビット長、 $n$  を  $P$  の格納に必要なビット長とする。 $0 \leq x \leq P * Q$  であるので、必ず  $m \geq n$  となる。まず、 $U = 2^m U_{\text{SQR}} = 2^{(2n)} U \bmod P$  を計算する(5030)。 $U_{\text{SQR}} = 2^{(2n)} U \bmod P$  を計算する部分の詳細な処理フローを図7に示す。図7では、 $2^L U \bmod P$  の計算手順が示されているが、 $L=2n$  として、図7の処理を用いることができる。 $U_{\text{SQR}}$  のビット長は、 $m-2n$  もしくは  $n$  のうちの長い方のビット長になる。5040の計算は、

$$A_R = (x * U_{\text{SQR}} + M * p) / 2^m \quad (\text{式3})$$

となり、 $x < 2^m$ 、 $M < 2^m$  であるので、

$$A_R < U_{\text{SQR}} + p \quad (\text{式4})$$

となる。 $p$  のビット長は  $n$  以下であるので、 $A_R$  のビット長は、 $\text{MAX}(m-2n, n)$  となる。このビット長が  $n$  以下となるには、

$$m-2n < n \quad (\text{式5})$$

$$m < 3n \quad (\text{式6})$$

である必要がある。通常の場合、 $m \approx 2n < 3n$  であるので、 $A_R$  のビット長は  $n$  となる。5050の処理を行うには、5040の処理結果が  $n$  以下となる必要がある。 $m < 3n$  の条件を満たさない場合には、図6に示す別の実施例による方法を行う。5050の処理を別の式で書くと、

$$(A_R + (-A_R * p^{(-1) \bmod 2^n}) * p) / 2^n \quad (\text{式7})$$

であるが、 $A_R < 2^n$  かつ  $(-A_R * p^{(-1) \bmod 2^n}) < 2^n$  であるので、

$$(A_R + (-A_R * p^{(-1) \bmod 2^n}) * p) / 2^n < 1+p \quad (\text{式8})$$

となり、必ず5050の処理の後の  $A_R$  は  $p$ 以下となり、 $n$ ビット長で表現できる。また、5050で  $A_R$ の値が  $p$ と等しくなるのは、 $x$ の値が  $p$ の倍数となる場合のみである。

5030から5050までの処理を数式で表現すると、

$$A_R \equiv x * 2^{(2n)} * 2^{(m)} * 2^{(-m)} * 2^{(-n)} \bmod P \quad (\text{式9})$$

$$\equiv x * 2^{(2n+m-m-n)} \bmod P \quad (\text{式10})$$

$$\equiv x * 2^n \bmod P \quad (\text{式11})$$

となる。

5050以降の処理は、図3の3050以降の処理と同一である。 $x$ の値が  $p$ の倍数となっている場合、図2中の2130、2140の処理にて最終的に補正される。

#### 【0013】

図6はモンゴメリ剰余乗算を用いた場合の本発明の別の一実施例を示す。 $m$ を入力  $x$ の格納に必要なビット長、 $n$ を  $P$ の格納に必要なビット長とする。 $0 \leq x \leq P * Q$ であるので、必ず  $m \geq n$ となる。まず、 $L=n+m$ として、図7に示されるフローに従って、 $U\_SQR=2^{(n+m)}U \bmod P$ を計算する(6030)。6040以降の処理を行うためには  $U\_SQR$ のビット数は  $m$ 以下である必要があるが、 $U\_SQR$ は必ず  $m$ ビット以下になる。6050の処理では、

$$(A_R + (-A_R * p^{(-1) \bmod 2^m}) * p) / 2^m \quad (\text{式12})$$

であるが、 $A_R < 2^n$  かつ  $(-A_R * p^{(-1) \bmod 2^m}) < 2^m$  であるので、

$$(A_R + (-A_R * p^{(-1) \bmod 2^m}) * p) / 2^m < 1+p \quad (\text{式13})$$

となり、必ず6050の処理の後の  $A_R$  は  $p$ 以下となり、 $n$ ビット長で表現できる。また、6050で  $A_R$ の値が  $p$ と等しくなるのは、 $x$ の値が  $p$ の倍数となる場合

のみである。

6030から6050までの処理を1つの式で表現すると、

$$A\_R \equiv x * 2^{(n+m)} * 2^{(m)} * 2^{(-m)} * 2^{(-m)} \bmod P \quad (\text{式14})$$

$$\equiv x * 2^{(n+m+m-m-m)} \bmod P \quad (\text{式15})$$

$$\equiv x * 2^n \bmod P \quad (\text{式16})$$

となる。

6050以降の処理は、図3の3050以降の処理と同一である。xの値がpの倍数となっている場合、図2中の2130、2140の処理にて最終的に補正される。また、図5の実施例と異なり、 $m < 3n$  という条件は必要ない。

#### 【0014】

図5、図6に示される実施例に必要な、 $2^L * U \bmod P$ を計算する実施例を図7に示す。図7の手順は下位ビットからのアディションチェーンにより、 $W := 2^L * R \bmod P$ を計算している。Wの初期値は、mビットに収まる形で、 $w \equiv 2 * (2^m) \bmod P$ となる値にする。Lを2進数表現したときに、最上位以外のビットに1がない場合は、L回の剰余自乗計算のみで計算できるが、L最上位ビット以外に1となるビットが有る場合には、途中で乗算を行う必要がある。従って、L最上位以外のビット位置に1が見つかったか否かを示す変数mulを用意しておく(7005)。7010の処理は、Pの最上位ビットがmの最上位ビットの位置と等しくなるようにシフトをした値を減じて、mビットの値に収まるようにするための処理である。7020、7030、7040、7050の処理は、Wの最上位2ビットを0とするための処理である。この処理は、最終的な演算結果がnビットに収まるようにするために行っている。処理が最上位ビットに達した際のチェックを行い(7060)達している場合は、変数mulの値をチェックし(7080)、1であれば、YにLの途中のビットに相当する結果が格納されているので、WにYを乗じ(7090)結果とする。処理が最上位ビットにまだ達していない場合は、Lの最下位ビット位置に1があるかチェックを行い(7070)見つかった場合は変数YにWの値を保存する。mulの値をチェックし(7100)、最上位以外のビットで1がはじめて見つかった場合は、YにWの値を代入し(7120)、mulに1を設定する(7130)。7120の処理は、本来Yに

1を代入しておき $Y := Y * W$ と計算することと等しい。また、7100のチェックで、すでに最上位以外のビットで1が見つまっている場合は、Yの値に現在のWの値を乗じる(7110)。Rの剰余自乗計算をモンゴメリ剰余乗算で行い(7140)、Lを1ビット右にシフトし(7150)、7060からの処理を繰り返す。

7140の $Y * Y * 2^{(-m)} \bmod p$ の計算は、

$$(A * A + M * p) / (2^m) \quad (\text{式17})$$

と等しい。ここで、

$$M = -Y * Y * (p^{(-1)}) \bmod 2^m < 2^m \quad (\text{式18})$$

である。

したがって、

$$(Y * Y + M * p) / (2^m) < (Y * Y + 2^m * p) / (2^m) \quad (\text{式19})$$

また、Yをmビット長のメモリに格納したときの、最上位の0となるビットの数をsとすると、

$$Y < 2^{(m-s)} \quad (\text{式20})$$

$$Y < 2^{(m-s)-1} \quad (\text{式21})$$

$$(Y * Y + 2^m * p) / (2^m) < ((2^{(m-s)-1})^2 + 2^m * p) / 2^m \quad (\text{式22})$$

$$= 2^{(m-2s)} - 2^{(1-s)} + 2^{(-m)} + p \quad (\text{式23})$$

$$< 2^{(m-2s)} + p + 1 \quad (\text{式24})$$

であるので、7140の $(Y * Y + M * p) / 2^m$ の演算結果は、1回演算するごとに最上位の0であるビットの数が、 $s[t+1] := 2s[t] - 1$ となる。従って、 $2^{(m-2s)}$ がpよりも大の場合は、t回計算したあと最上位から連続する0のビット数は、 $s[0] * 2^{(t-1)}$ 個となり、 $2^{(m-2s)}$ がpよりも小さくなれば、ビット長はpによって決定される。8020、8030、8040、8050の処理で最上位の2ビットは0に設定されるので、 $s[0] = 2$ となり、t回8140を実行した後のビット数は、 $m - 2^t$ もしくはnとなる。また、tの回数は、 $t = \log_2(L)$ であるので、ビット数は、 $\max(m - L, n)$ となる。

【0015】

図8は通常の剰余乗算を用いた場合の本発明の一実施例を示す。mを入力xの

格納に必要なビット長、 $n$ を $P$ の格納に必要なビット長とする。まず、図9のフローに従って、 $2^n \bmod P$ を計算し、 $R$ とする(8020)。次に、入力 $x$ に $R$ を乗じる(8030)。最後に補正を行うための $R\_ITOTAL$ を計算する(8040)が、この処理は $P$ が確定していれば、入力値 $x$ と独立して計算することが可能であるので、予め計算して保存して置いても良い。実際の計算では、まず $2^{(-n)} \bmod p$ を「図10」の手順に従って計算し、 $R\_INV$ とし、 $R$ および $R\_INV$ を元に図11の手順に従って、 $(R\_INV) \cdot (2^{n-1}) \bmod P$ を計算し、 $R\_ITOTAL$ とする。通常の剰余乗算を用いるので、演算の初期値 $W$ は1とし、カウンタ $i$ に最上位ビットの位置を示す $n-1$ を設定する(8050)。指数上位ビットから順に1が立っているかどうかを調べ(8060)、1が立っていれば、 $xR \bmod P (=A\_R)$ を $W$ に乘じ(8080)、ビットの値が0であれば、1を意味する $R$ を乗じる(8070)。ビットの位置を1ビット下に移動し(8090)、最下位ビットまで処理が終了したかをチェックし(8100)、最下位ビットまで処理がたどり着いていない場合は、結果を二乗し(8110)、指数の次のビットの処理を行う。8070、8080の処理では通常の処理と比べると必ず $R$ が毎回余計に乘じられるため、最下位ビットまで処理が終了した場合は、 $R \cdot (2^{n-1})$ が余分に乘じられることになる。余分に乘じられた $R \cdot (2^{n-1})$ の影響を取り除くために、最後に $R\_ITOTAL$ (8210)を乗じる。

#### 【0016】

図9は、図8の実施例中の8020で $2^L \bmod P$ を計算する処理の一実施例である。図9の手順は下位ビットからのアディションチェーンにより、 $R := 2^L \bmod P$ を計算している。 $L$ を2進数表現したときに、最上位以外のビットに1がない場合は、 $L$ 回の剰余自乗計算のみで計算できるが、 $L$ 最上位ビット以外に1となるビットが有る場合には、途中で乗算を行う必要がある。 $L$ の最上位以外のビット位置に1が見つかったか否かを示す変数 $mul$ を用意し、0に初期化しておく(9005)。次に $R$ を2で初期化する(9010)。処理が最上位ビットに達した際のチェックを行い(9060)達している場合は、変数 $mul$ の値をチェックし(9080)、1であれば、 $Y$ に $L$ の途中のビットに相当する結果が格納されているので、 $R$ に $Y$ を乗じ(9090)結果とする。最上位ビットにまで処理

が達していない場合は、Lの最下位ビット位置に1があるかチェックを行い（9070）見つかった場合は変数Yに保存する。mulの値をチェックし（9100）、最上位以外のビットで1がはじめて見つかった場合は、YにRの値を代入し（9120）、mulに1を設定する（9130）。9120の処理は、本来Yに1を代入しておき $Y := Y * R$ と計算することと等しい。また、9100のチェックで、すでに過去に最上位以外のビットで1が見つかった場合は、Yの値に現在のRの値を乗じる（9110）。Rの剰余自乗を計算し（9140）、Lを1ビット分右シフトしたのち（9150）、再び9060からの処理に戻る。

【0017】

図10は、図8の実施例中の8040で $R\_INV := 2^{(-n)} \bmod P$ を計算する処理の一実施例である。図10の手順は下位ビットからのアディションチェーンにより、 $R\_INV := 2^{(-L)} \bmod P$ を計算している。Lを2進数表現したときに、最上位以外のビットに1がない場合は、L回の剰余自乗計算のみで計算できるが、L最上位ビット以外に1となるビットが有る場合には、途中で乗算を行う必要がある。Lの最上位以外のビット位置に1が見つかったか否かを示す変数mulを用意し、0に初期化しておく（10005）。次にRを $1/2$ で初期化する（10010）。 $1/2$ に初期化するには、1を1回右シフトすればよいが、1を右シフトすると0になるため、まずPを加えてから右シフトを行う。Pの値は大きな素数であるので、必ず奇数となる。したがって、 $1+P$ は必ず偶数になるため、右シフトが可能である。つぎに処理が最上位ビットに達した際のチェックを行い（10060）達している場合は、変数mulの値をチェックし（10080）、1であれば、YにLの途中のビットに相当する結果が格納されているので、 $R\_INV$ にYを乗じ（10090）結果とする。最上位ビットにまで処理が達していない場合は、最上位以外のビット位置に1があるかチェックを行い（10070）見つかった場合は変数Yに保存する。mulの値をチェックし（9100）、最上位以外のビットで1がはじめて見つかった場合は、YにRの値を代入し（9120）、mulに1を設定する（9130）。9120の処理は、本来Yに1を代入しておき $Y := Y * R$ と計算することと等しい。また、9100のチェックで、すでに過去に最上位以外のビットで1が見つかった場合は、Yの値に現在のR

の値を乗じる(9110)。Rの剰余自乗を計算し(9140)、Lを1ビット分右シフトしたのち(9150)、再び9060からの処理に戻る。

【0018】

図11は、図8の実施例中の8040で $R\_ITOTAL := (R\_INV)^{(2^n-1)} \bmod P$ を計算する処理の一実施例である。演算方法は、式26に示すように、 $R\_INV^{(2^n)}$ にRを乗ずることにより行う。 $R\_INV^{(2^n)}$ は、剰余自乗演算をn回繰り返すことにより計算する。

$$(R\_INV)^{(2^n-1)} \bmod P = R\_INV^{(2^n)} * R\_INV^{(-1)} \bmod P \quad (\text{式25})$$

$$= R\_INV^{(2^n)} * R \bmod P \quad (\text{式26})$$

まず、 $R\_ITOTAL$ を $R\_INV$ で初期化し(11010)、つぎに剰余自乗算を行う回数nを変数iに代入(11020)、 $R\_ITOTAL$ の剰余自乗した値を $R\_ITOTAL$ に代入し(11030)、カウンタ用変数iから1を減じ(11040)、カウンタ変数iの値が0よりも大きければ(11050)、11030からの処理を繰り返す。最後にRを $R\_ITOTAL$ に剰余乗算した値を $R\_ITOTAL$ に代入し(11060)、結果として返す。

付記：

1. 法N上での計算において、非演算数にあらかじめNと互いに素な値をべき乗した値を乗じ、計算後に前記Nと互いに素な値のべき乗した値の法Nに対する逆元を乗ずることで計算することを特徴とした処理方法。
2. 法N上での剰余計算において、非演算数にあらかじめNと互いに素な値をべき乗した値を乗じ、計算後に前記Nと互いに素な値のべき乗した値の法Nに対する逆元を乗ずることを特徴とした処理方法において、法Nが2よりも大きな素数の積であり、前記Nと互いに素である値として2を用いることを特徴とした処理方法。
3. モンゴメリ剰余乗算装置を有し、Pを素数とし、 $x > P$ であるxに対して、 $x * (2^n) \bmod P$ を計算する情報処理装置に於いて、法をP、入力値をx、法Pを格納するために必要十分なビット数をn、入力値xを格納するために必要なビット長をmとし、入力値を $x * (2^n) \bmod P$ に変換する際に、 $2^{(2m+n)} \bmod P$ を計算するか、あるいは予め用意しておき、モンゴメリ剰余乗算



装置により  $x_1 = x * 2^{(2m+n)} * (2^{(-m)}) \bmod P = x * 2^{(m+n)} \bmod P$  を計算し、さらに  $x_2 := x_1 * (2^{(-m)}) \bmod P = x * (2^n) \bmod P$  を計算することにより、 $x \bmod P$  を陽に求めること無しに  $x * (2^n) \bmod P$  を計算することを特徴とする情報処理装置。

4.  $P$  を素数とし、 $x > P$  である  $x$  に対して、 $x * (2^n) \bmod P$  を計算する情報処理方法に於いて、法を  $P$ 、入力値を  $x$ 、法  $P$  を格納するために必要十分なビット数を  $n$ 、入力値  $x$  を格納するために必要なビット長を  $m$  とし、入力値を  $x * (2^n) \bmod P$  に変換する際に、 $2^{(2m+n)} \bmod P$  を計算するか、あるいは予め用意しておき、モンゴメリ剰余乗算により  $x_1 = x * 2^{(2m+n)} * (2^{(-m)}) \bmod P = x * 2^{(m+n)} \bmod P$  を計算し、さらに  $x_2 := x_1 * (2^{(-m)}) \bmod P = x * (2^n) \bmod P$  を計算することにより、 $x \bmod P$  を陽に求めること無しに  $x * (2^n) \bmod P$  を計算することを特徴とする情報処理方法。

5. モンゴメリ剰余乗算装置を有し、 $P$  を素数とし、 $x > P$  である  $x$  に対して、 $x * (2^n) \bmod P$  を計算する情報処理装置に於いて、法を  $P$ 、入力値を  $x$ 、法  $P$  を格納するために必要十分なビット数を  $n$ 、入力値  $x$  を格納するために必要なビット長を  $m$  とし、入力値を  $x * (2^n) \bmod P$  に変換する際に、 $2^{(m+2n)} \bmod P$  を計算するか、あるいは予め用意しておき、モンゴメリ剰余乗算装置により  $x_1 = x * 2^{(m+2n)} * (2^{(-m)}) \bmod P = x * 2^{(2n)} \bmod P$  を計算し、さらに  $x_2 := x_1 * (2^{(-n)}) \bmod P = x * (2^n) \bmod P$  を計算することにより、 $x \bmod P$  を陽に求めること無しに  $x * (2^n) \bmod P$  を計算することを特徴とする情報処理装置。

6.  $P$  を素数とし、 $x > P$  である  $x$  に対して、 $x * (2^n) \bmod P$  を計算する情報処理方法に於いて、法を  $P$ 、入力値を  $x$ 、法  $P$  を格納するために必要十分なビット数を  $n$ 、入力値  $x$  を格納するために必要なビット長を  $m$  とし、入力値を  $x * (2^n) \bmod P$  に変換する際に、 $2^{(m+2n)} \bmod P$  を計算するか、あるいは予め用意しておき、モンゴメリ剰余乗算方式により  $x_1 = x * 2^{(m+2n)} * (2^{(-m)}) \bmod P = x * 2^{(2n)} \bmod P$  を計算し、さらに  $x_2 := x_1 * (2^{(-n)}) \bmod P = x * (2^n) \bmod P$  を計算することによ

り、 $x \bmod P$ を陽に求めること無しに $x * (2^n) \bmod P$ を計算することを特徴とする情報処理方法。

7. 法をP、入力値をx、法Pを格納するために必要十分なビット数をn、入力値xを格納するために必要なビット長をm、指数をdとし、べき乗の指数をsビットづつとりだしてsビット毎のべき乗演算の結果を合成して、 $x^d \bmod P$ のべき乗剰余演算を行う情報処理装置において、sビットづつ取り出した指数のi番目の指数をd[i]としたとき、 $x^{d[i] \bmod P}$ を演算する代わりに $(2^n)^{(2^{s-1})} * x^{d[i] \bmod P}$ を用いて演算を行い、 $(2^n)^{(2^{n-1})} * x^d \bmod P$ を計算した後、 $2^{(-n)} * (2^{n-1}) \bmod P$ を乗じて、 $x^d \bmod P$ を計算することを特徴とする情報処理装置。

8. 法をP、入力値をx、法Pを格納するために必要十分なビット数をn、入力値xを格納するために必要なビット長をm、指数をdとし、べき乗の指数をsビットづつとりだしてsビット毎のべき乗演算の結果を合成して、 $x^d \bmod P$ のべき乗剰余演算を行う情報処理方法において、sビットづつ取り出した指数のi番目の指数をd[i]としたとき、 $x^{d[i] \bmod P}$ を演算する代わりに $(2^n)^{(2^{s-1})} * x^{d[i] \bmod P}$ を用いて演算を行い、 $(2^n)^{(2^{n-1})} * x^d \bmod P$ を計算した後、 $2^{(-n)} * (2^{n-1}) \bmod P$ を乗じて、 $x^d \bmod P$ を計算することを特徴とする情報処理方法。

【0019】

#### 【発明の効果】

本発明によれば、べき乗剰余演算のCRT計算において、秘密素数による入力値の剰余計算を直接行わずに計算できるため、入力を変えながら、消費電流などを観測することで秘密素数を推定することが困難となる。「図12」は通例の方法による計算時の $x \bmod P$ のビット長およびハミングウェイト（値を2進数表現した場合に1になっているビットの数）を示す。「図13」は本発明における、 $x \bmod P$ に相当する $x \cdot 2^n \bmod P$ のビット長及びハミングウェイトを示す。図12では入力データの値と $x \bmod P$ の間に明らかな

依存性が現れているが、図13では入力データに依存せず、ビット長およびハミングウェイトが一定値となり、依存性が現れないことが確認できる。

【図面の簡単な説明】

【図1】

通例のRSA用CRT演算方式の処理フロー。

【図2】

モンゴメリ剰余乗算を用いた場合の、通例のCRT方式向けべき乗剰余演算の処理のフロー。

【図3】

$x$  と  $x$  を秘密素数  $p$  で剰余計算した結果のグラフ。

【図4】

通常の剰余乗算を用いた場合の、通例のCRT方式向けべき乗剰余演算の処理のフロー。

【図5】

本発明による、モンゴメリ剰余乗算を用いたセキュアな剰余演算処理の一実施例。

【図6】

本発明による、モンゴメリ剰余乗算を用いたセキュアな剰余演算処理の別の一実施例。

【図7】

本発明による、モンゴメリ剰余乗算を用いたセキュアな剰余演算処理の部分処理一実施例。

【図8】

本発明による、セキュアなべき乗剰余演算処理の一実施例。

【図9】

本発明による、セキュアなべき乗剰余演算処理の部分処理の一実施例。

【図10】

本発明による、セキュアなべき乗剰余演算処理の部分処理の一実施例。

【図11】

本発明による、セキュアなべき乗剰余演算処理の部分処理の一実施例。

【図 1 2】

通例の  $x$  と  $x$  を秘密素数  $p$  で剰余計算した結果のグラフ。

【図 1 3】

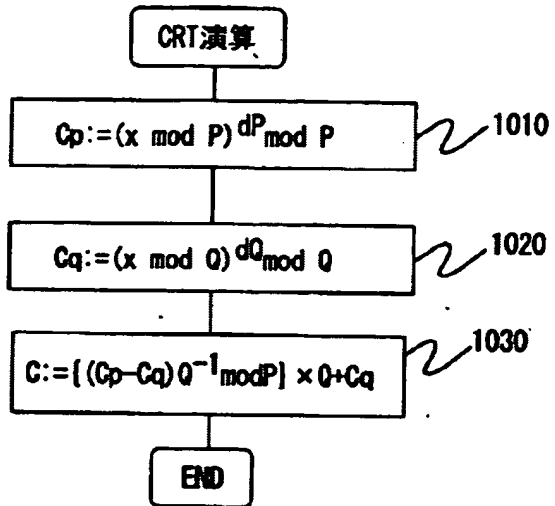
本発明による  $x$  と  $x$  を秘密素数  $p$  で剰余計算した結果のグラフ。

【書類名】

図面

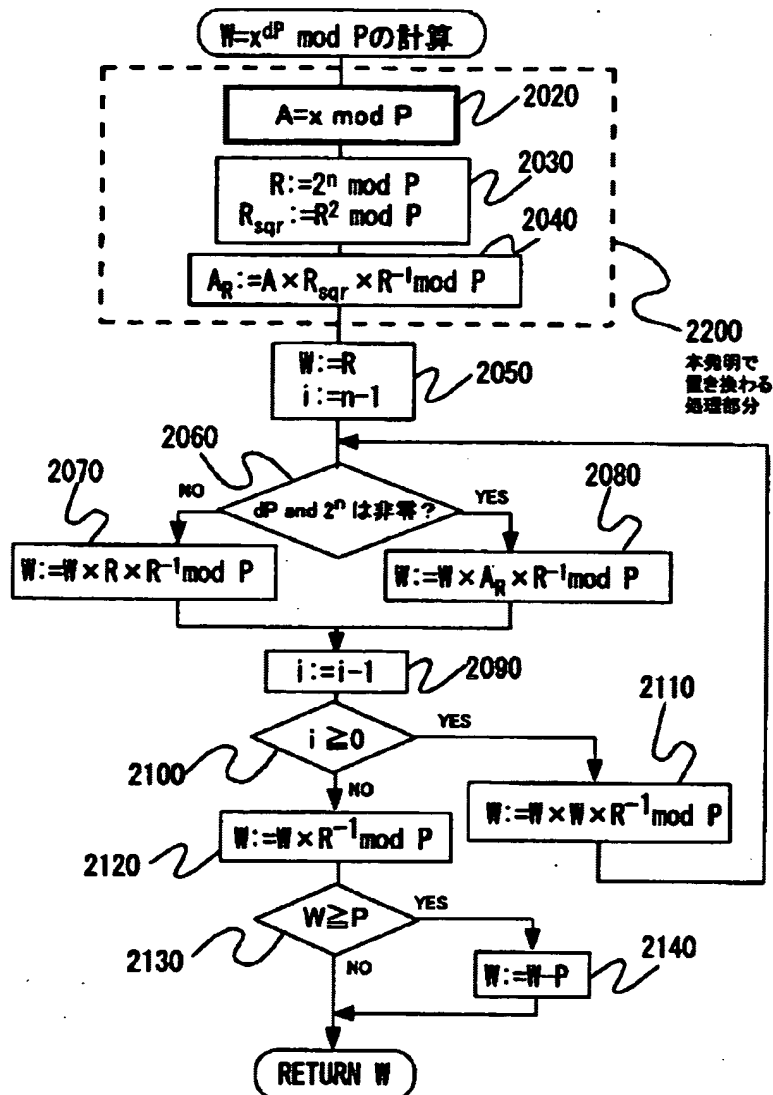
【図1】

図 1



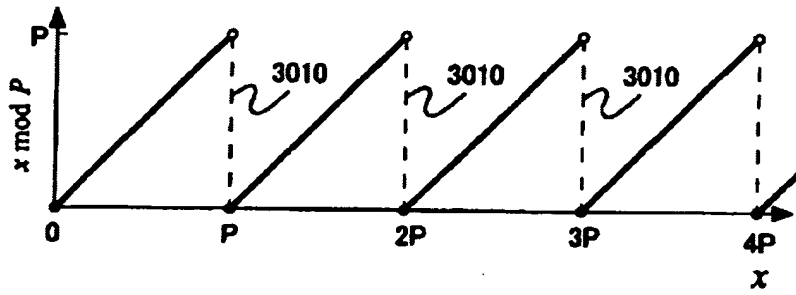
【図2】

図2



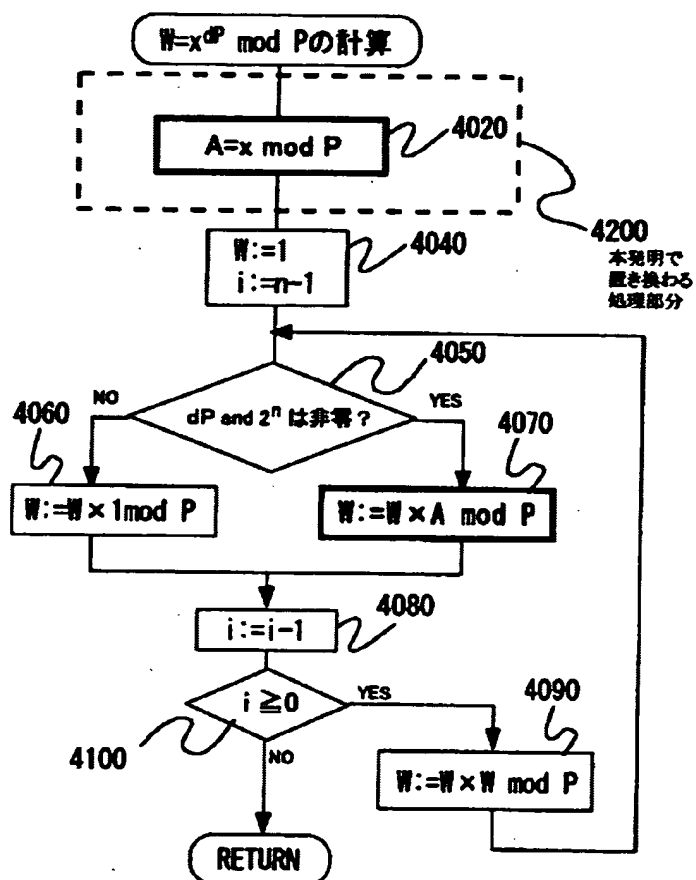
【図3】

図3



【図4】

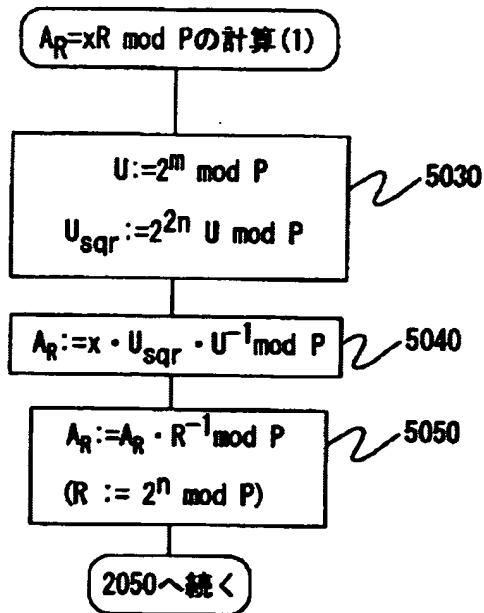
図4





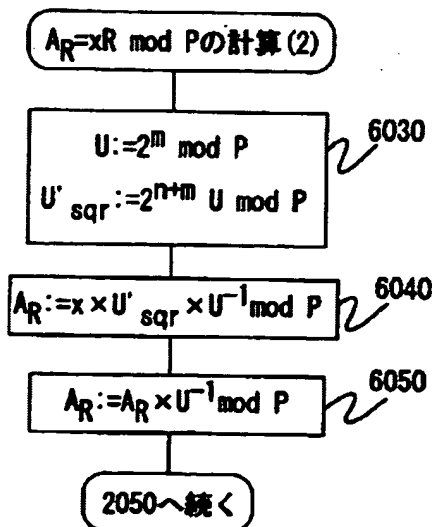
【図 5】

図 5



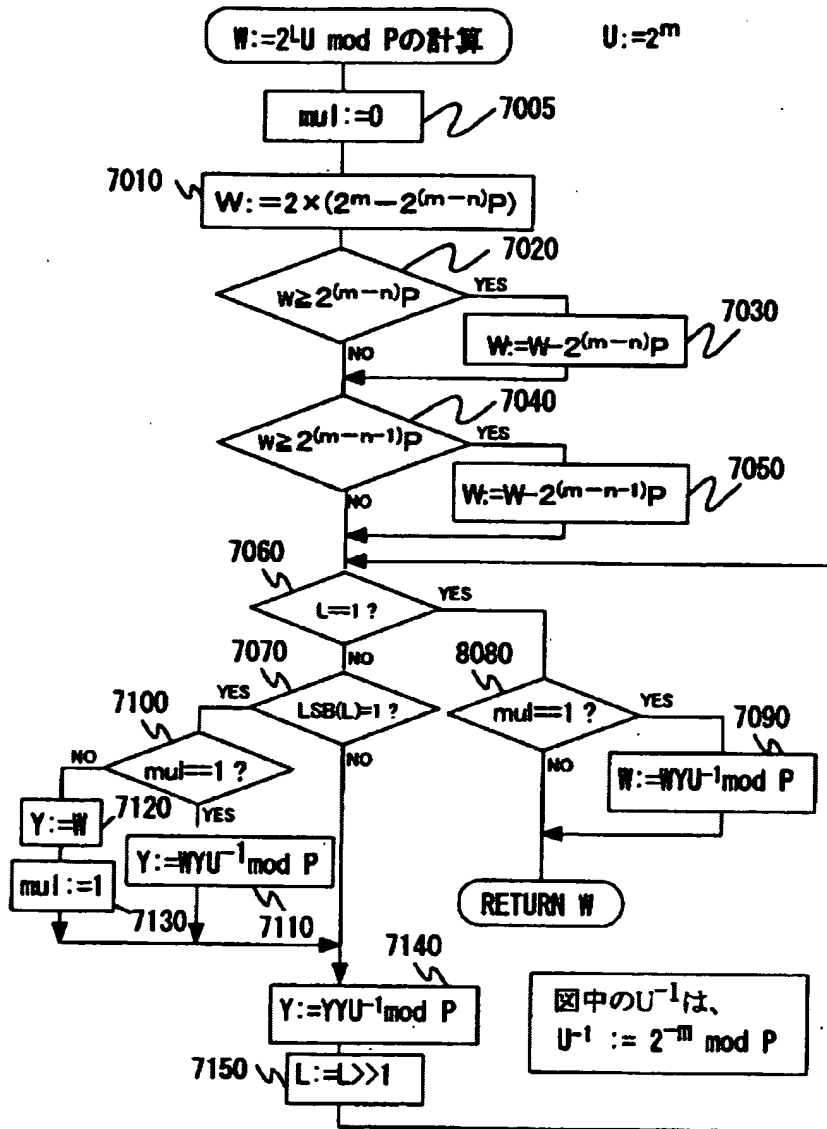
【図 6】

図 6



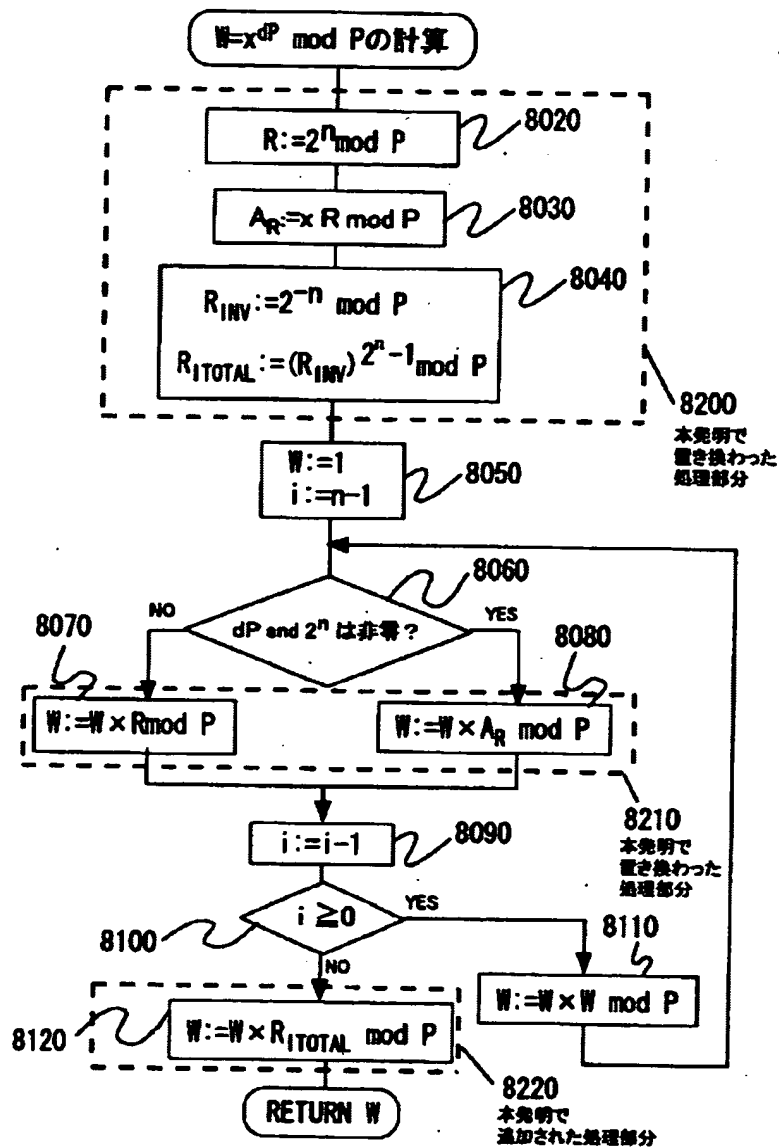
【図7】

図7



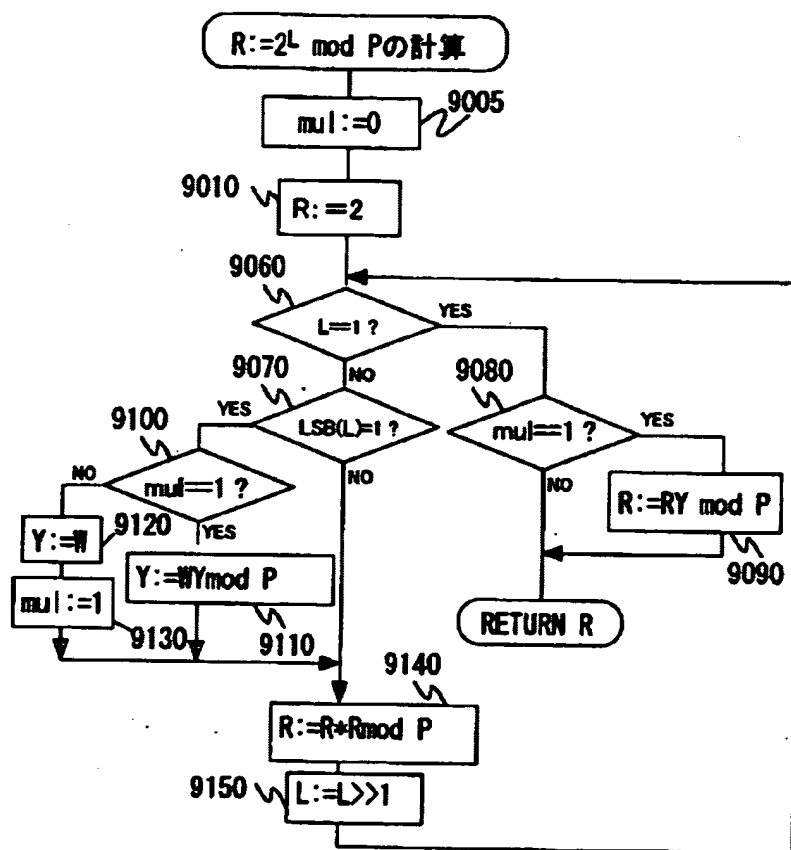
【図 8】

図 8



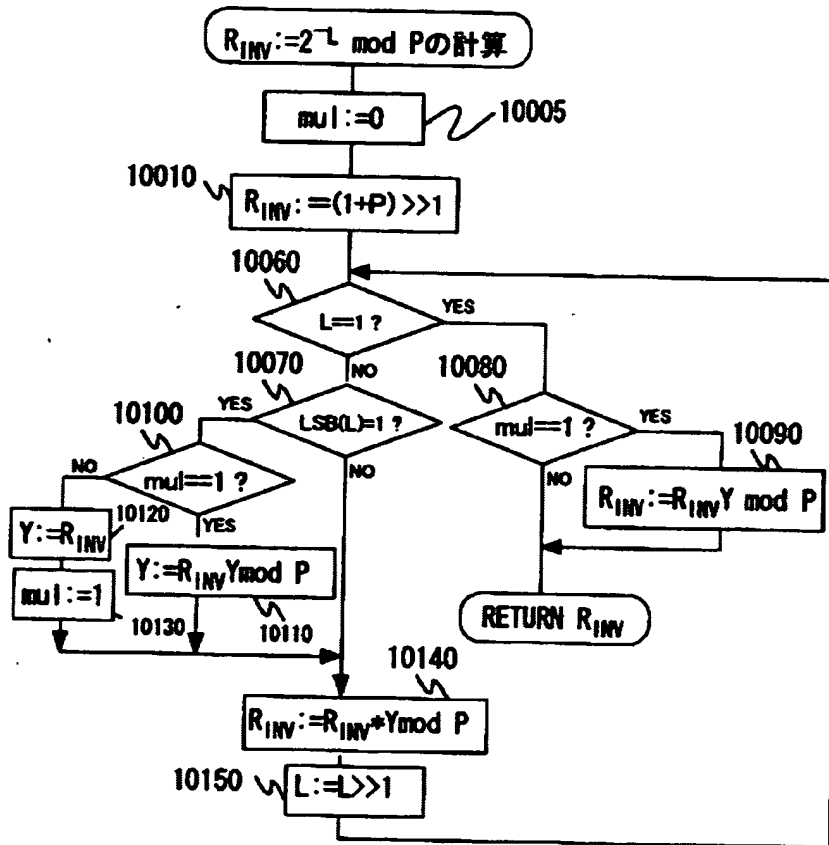
【図9】

図9



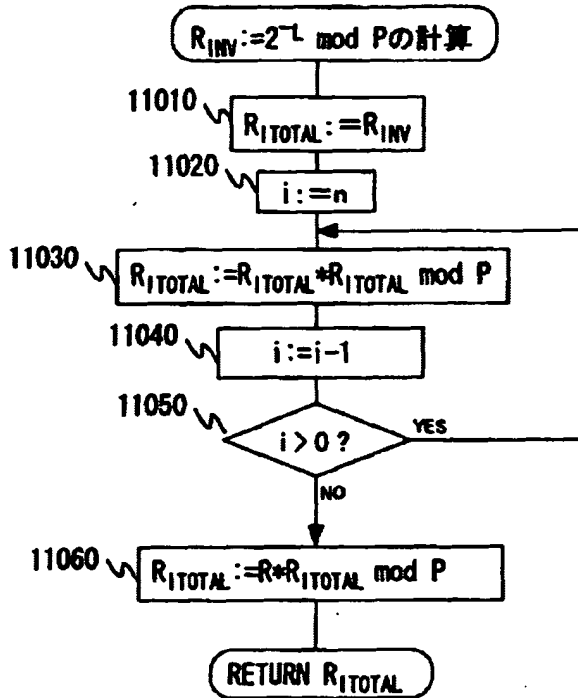
【図10】

図10



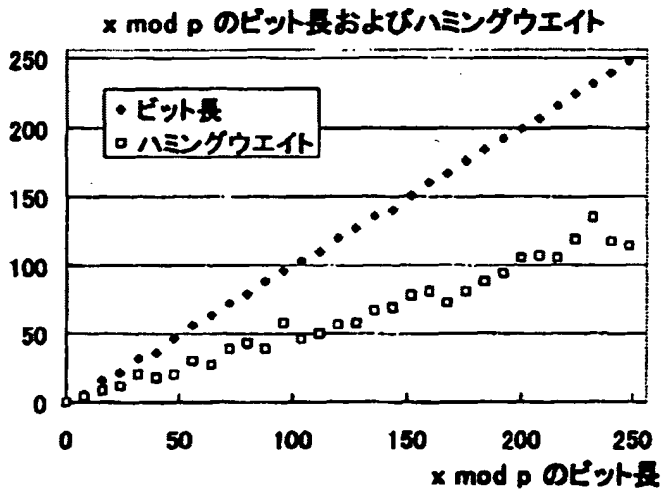
【図11】

図11



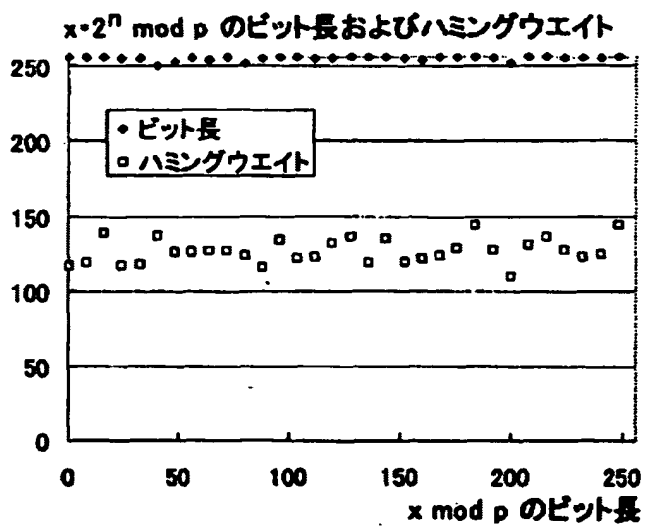
【図12】

図12



【図 13】

図 13



【書類名】 要約書

【要約】

【課題】 RSA暗号の高速演算手法として、中国人剰余定理を用いた計算手法が広く用いられているが、最初の演算で、秘密素数 $p$ による剰余計算が必要となる。この計算は、秘密素数 $p$ を陽に用いた計算であるので、古くからアタックの対象となっている。

【解決手段】  $x \bmod p$  を直接計算せずに、「図5」に示すように $2^{(m+n)} \bmod p$  もしくは、 $2^{(2n)} \bmod p$  をあらかじめ $x$ に乗じておき、その後、 $2^{(-n)}$  もしくは、 $2^{(-m)}$  を乗じて、 $2^n x \bmod p$  を計算する。モンゴメリ剰余乗算を用いる場合は、その後の処理は通例通りとなる。通常の剰余乗算を用いる場合は、べき乗剰余演算の最後に、 $(2^{(-n)})^{(2^n-1)} \bmod p$  を乗じ補正する。

【選択図】 図5



認定・付加情報

特許出願の番号	特願2003-014136
受付番号	50300100380
書類名	特許願
担当官	第八担当上席 0097
作成日	平成15年 1月24日

<認定情報・付加情報>

【提出日】	平成15年 1月23日
-------	-------------

出願人履歴情報

識別番号 [000005108]

1. 変更年月日 1990年 8月31日  
[変更理由] 新規登録  
住 所 東京都千代田区神田駿河台4丁目6番地  
氏 名 株式会社日立製作所